NAME

intro — introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, etc.

NAME

ascii – map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dc1	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 $	045 %	046 &	047 '
050 (	051)	052 *	053 +	054 ,	055 –	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174 |	175 }	176 ~	177 del

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d –	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

FILES

/usr/pub/ascii

NAME

diskcnfg – system disk drive configuration

DESCRIPTION

Diskcnfg contains information pertaining to the disk and tape configuration of the system. Its primary purpose is to direct the actions of *filesave* (1M).

The format of the file consists of four basic sections, each delimited by a line beginning with "$". Each section is comprised of lines containing one or more fields separated by white space. Null lines and lines beginning with "*" are intended to be ignored.

Section one describes the configuration and type of all disk and tape units configured to the system. Lines in this section have the format:

keyword #DR type

where *keyword* is either **DRIVE** or **SPARE**, *DR#* is the drive number, and *type* indicates the type of drive. Valid specifications for *type* are:

disk	*tape*
rp03	**tu10**
rp04	**tu16**
rp05	**tu16d**
rp06	
rm05	

Type is assumed **rp04** if not specified.

Section two describes the file system configuration. Lines in this section have the format:

fsname section pkname

where *fsname* is a file system name, *section* is the drive and section number, and *pkname* is the volume label.

Section three contains lists of spare volumes to be used in file saves. The format of this section is:

day name1 name2...

where *day* is the day of the week (0–6, with 0=sunday), and *name1...* comprising the list of spare volume names.

Section four describes the file save procedure. Lines in this section contain from 1–8 fields (sections of a save pack) outlining the order in which file systems from section two are to be placed on the spare volumes. A field name of

"—" indicates that the corresponding section of the spare pack is not to be used.

EXAMPLE

The following illustration is an example of a valid *diskcnfg* file.

```
*       Disk Configuration Section.

DRIVE 0     rp06
DRIVE 1     rp06
DRIVE 2     rp06
DRIVE 3     rp06

SPARE 2     rp06

$$$

*       Filesystem Configuration.

root    0       pk1
usr     1       pk1
a1      31      pk2
a4      33      pk2
a2      11      pk5
a3      13      pk5

$$$

*       Spare Pack List.

0
1       pk7     pk12
2       pk10    pk6
3       pk11    pk8
4       pk2     pk3
5       pk14    pk9
6

$$$

*       File Save Configuration.

root    usr     a1      —       a4      —       —
—       a2      —       a3      —       —       —

$$$
```

SEE ALSO

filesave(1M).

NAME

environ – user environment

DESCRIPTION

An array of strings called the "environment" is made available by *exec*(2) when a process begins. By convention, these strings have the form "name=value". The following names are used by various commands:

PATH The sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). *Login*(1) sets **PATH=:/bin:/usr/bin**.

HOME Name of the user's login directory, set by *login* from the password file *passwd*(4).

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as *mm*(1) or *tplot*(1G), which may exploit special capabilities of that terminal.

TZ Time zone information. The format is **xxx***n***zzz** where **xxx** is standard local time zone abbreviation, *n* is the difference in hours from GMT, and **zzz** is the abbreviation for the daylight-saving local time zone, if any; for example, **EST5EDT**.

Further names may be placed in the environment by the *export* command and "name=value" arguments in *sh*, or by *exec*. It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL**, **PS1**, **PS2**, **IFS**.

SEE ALSO

env(1), login(1), mm(1), nice(1), nohup(1), sh(1), time(1), tplot(1G), exec(2), passwd(4).

NAME

eqnchar – special character definitions for eqn and neqn

SYNOPSIS

eqn /usr/pub/eqnchar [files] | **troff** [options]

neqn /usr/pub/eqnchar [files] | **nroff (1)** [options]

eqn -Taps /usr/pub/apseqnchar [files] | **troff** [options]

eqn -Tcat /usr/pub/cateqnchar [files] | **otroff** [options]

DESCRIPTION

Eqnchar contains *troff*(1) and *nroff*(1) character definitions for constructing characters that are not available on a phototypesetter. These definitions are primarily intended for use with *eqn*(1) and *neqn*; *eqnchar* contains definitions for the following characters:

ciplus	⊕	||	‖	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	∼	*rangle*	⟩	*blot*	■
−wig	≃	*hbar*	ℏ	*bullet*	•
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	*<−>*	↔	*empty*	∅
=wig	≅	*<=>*	⇔	*member*	∈
star	∗	*|<*	⊀	*nomem*	∉
bigstar	✳	*|>*	⊁	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	∟	*incl*	⊑
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
angstrom	Å	*degree*	°	*scrL*	ℓ
==<	≦	*==>*	≧		

Apseqnchar is a version of *eqnchar* tailored for the Autologic APS-5 phototypesetter. This will not look optimal on other phototypesetters. Similarly, *cateqnchar* is the old *eqnchar* tailored for the Wang CAT and the old *otroff*. Until a phototypesetter-independent version of *eqnchar* is available, *eqnchar* should be a link to the default version on each system. The standard default is *apseqnchar*.

FILES

/usr/pub/eqnchar
/usr/pub/apseqnchar
/usr/pub/cateqnchar

SEE ALSO

eqn(1), nroff(1), troff(1).
UNIX System User Reference Manual.

NAME

fcntl – file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

The *fcntl*(2) function provides for control over open files. This include file describes *requests* and *arguments* to *fcntl* and *open*(2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/*  (The first three can only be set by open) */
#define O_RDONLY  0
#define O_WRONLY  1
#define O_RDWR    2
#define O_NDELAY  04      /* Non-blocking I/O */
#define O_APPEND  010     /* append (writes guaranteed at the end) */

/* Flag values accessible only to open(2) */
#define O_CREAT   00400   /* open with file create (uses third open arg)*/
#define O_TRUNC   01000   /* open with truncation */
#define O_EXCL    02000   /* exclusive open */

/* fcntl(2) requests */
#define F_DUPFD   0       /* Duplicate fildes */
#define F_GETFD   1       /* Get fildes flags */
#define F_SETFD   2       /* Set fildes flags */
#define F_GETFL   3       /* Get file flags */
#define F_SETFL   4       /* Set file flags */
```

SEE ALSO

fcntl(2), open(2).

NAME

font – description files for device-independent troff

SYNOPSIS

troff –T*ptty* ...

DESCRIPTION

For each phototypesetter supported by *troff*(1) and available on this system, there is a directory containing files describing the device and its fonts. This directory is named **/usr/lib/font/dev***ptty* where *ptty* is the name of the phototypesetter. Currently the only *ptty* supported is **aps** for the Autologic APS–5.

For a particular phototypesetter, *ptty*, the ASCII file *DESC* in the directory **/usr/lib/font/dev***ptty* describes its characteristics. Each line starts with a word identifying the characteristic and followed by appropriate specifiers. Blank lines and lines beginning with a # are ignored.

The legal lines for *DESC* are:

res *num*	resolution of device in basic increments per inch
hor *num*	smallest unit of horizontal motion
vert *num*	smallest unit of vertical motion
unitwidth *num*	pointsize in which widths are specified
sizescale *num*	scaling for fractional pointsizes
paperwidth *num*	width of paper in basic increments
paperlength *num*	length of paper in basic increments
spare1 *num*	available for use
spare2 *num*	available for use
sizes *num num ...*	list of pointsizes available on typesetter
fonts *num name ...*	number of initial fonts followed by the names of the fonts. For example: fonts 4 R I B S
charset	this always comes last in the file and is on a line by itself. Following it is the list of special character names for this device. Names are separated by a space or a newline. The list can be as long as necessary. Names not in this list are not allowed in the font description files.

Res is the basic resolution of the device in increments per inch. **Hor** and **vert** describe the relationships between motions in the horizontal and vertical directions. If the device is capable of moving in single basic increments in both directions, both **hor** and **vert** would have values of 1. If the vertical motions only take place in multiples of two basic units while the horizontal motions take place in the basic increments, then **hor** would be 1, while **vert** would be 2. **Unitwidth** is the pointsize in which all width tables in the font description files are given. *Troff* automatically scales the widths from the **unitwidth** size to the pointsize it is working with. **Sizescale** is not currently used and is 1. **Paperwidth** is the width of the paper in basic increments. The APS-5 is 6120 increments wide. **Paperlength** is the length of a sheet of paper in the basic increments.

For each font supported by the phototypesetter, there is also an ASCII file with the same name as the font (e.g., **R, I, CW**). The format for a font description file is:

name *name*	name of the font, such as **R** or **CW**
internalname *name*	internal name of font
special	sets flag indicating that the font is special
ligatures *name* ... **0**	Sets flag indicating font has ligatures. The list of ligatures follows and is terminated by a zero. Accepted ligatures are: **ff fi fl ffi ffl**.
spare1	available for use
spacewidth *num*	width of space if something other than 1/3 of \(em is desired as a space.
charset	The charset must come at the end. Each line following the word *charset* describes one character in the font. Each line has one of two formats: *name width kerning code* *name* "

where *name* is either a single ASCII character or a special character name from the list found in *DESC*. The width is in basic increments. The kerning information is 1 if the character descends below the line, 2 if it rises above the letter 'a', and 3 if it both rises and descends. The kerning information for special characters is not used and so may be 0. The code is the number sent to the typesetter to produce the character. The second format is used to indicate that the character has more than one name. The double quote indicates that this name has the same values as the preceding line. The kerning and code fields are not used if the width field is a double quote character.

Troff and its postprocessors read this information from binary files produced from the ASCII files by a program distributed with *troff* called *makedev*. For those with a need to know, a description of the format of these files follows:

The file *DESC.out* starts with the *dev* structure, defined by *dev.h*:

```
 /*
dev.h: characteristics of a typesetter
* /

struct dev {
short     filesize;      /* number of bytes in file, */
                /* excluding dev part */
short     res;           /* basic resolution in goobies/inch */
short     hor;           /* goobies horizontally */
short     vert;
short     unitwidth;     /* size at which widths are given*/
short     nfonts;        /* number fonts physically available */
short     nsizes;        /* number of pointsizes */
short     sizescale;     /* scaling for fractional pointsizes */
short     paperwidth;    /* max line length in units */
short     paperlength;    /* max paper length in units */
short     nchtab;        /* number of funny names in chtab */
short     lchname;       /* length of chname table */
short     spare1;        /* in case of expansion */
short     spare2;
};
```

Filesize is just the size of everything in *DESC.out* excluding the *dev* structure. *Nfonts* is the number of different font positions available. *Nsizes* is the number of different pointsizes supported by this typesetter. *Nchtab* is the number of special character names. *Lchname* is the total number of characters, including nulls, needed to list all the special character names. At the end of the structure are two spares for later expansions.

Immediately following the *dev* structure are a number of tables. First is the *sizes* table, which contains *nsizes* + 1 shorts(a null at the end), describing the pointsizes of text available on this device. The second table is the *funny_char_index_table*. It contains indices into the the table which follows it, the *funny_char_strings*. The indices point to the beginning of each special character name which is stored in the *funny_char_strings* table. The *funny_char_strings* table is *lchname* characters long, while the *funny_char_index_table* is *nchtab* shorts long.

Following the *dev* structure will occur *nfonts* {*font*}*.out* files, which are used to initialize the font positions. These {*font*}*.out* files, which also exist as separate files, begin with a *font* structure and then are followed by four character arrays:

```
struct font {       /* characteristics of a font */
char nwfont;       /* number of width entries */
char specfont;     /* 1 == special font */
char  ligfont;     /* 1 == ligatures exist on this font */
char spare1;       /* unused for now */
char namefont[10]; /* name of this font, e.g., R */
char intname[10];  /* internal name of font, in ASCII */
};
```

The *font* structure tells how many defined characters there are in the font, whether the font is a "special" font and if it contains ligatures. It also has the ASCII name of the font, which should match the name of the file it appears in, and the internal name of the font on the typesetting device (*intname*). The internal name is independent of the font position and name that *troff* knows about. For example, you might say mount R in position 4, but when asking the typesetter to actually produce a character from the R font, the postprocessor which instructs the typesetter would use *intname*.

The first three character arrays are specific for the font and run in parallel. The first array, *widths*, contains the width of each character relative to *unitwidth*. *Unitwidth* is defined in *DESC*. The second array, *kerning*, contains kerning information. If a character rises above the letter 'a', 02 is set. If it descends below the line, 01 is set. The third array, *codes*, contains the code that is sent to the typesetter to produce the character.

The fourth array is defined by the device description in *DESC*. It is the *font_index_table*. This table contains indices into the *width*, *kerning*, and *code* tables for each character. The order that characters appear in these three tables is arbitrary and changes from one font to the next. In order for *troff* to be able to translate from ASCII and the special character names to these arbitrary tables, the *font_index_table* is created with an order which is constant for each device. The number of entries in this table is 96 plus the number of special character names for this device. The value 96 is 128 - 32, the number of printable characters in the ASCII alphabet. To determine whether a

normal ASCII character exists, *troff* takes the ASCII value of the character, subtracts 32, and looks in the *font_index_table*. If it finds a 0, the character is not defined in this font. If it finds anything else, that is the index into *widths*, *kerning*, and *codes* that describe that character.

To look up a special character name, for example **\(pl**, the mathematical plus sign, and determine whether it appears in a particular font or not, the following procedure is followed. A *counter* is set to 0 and an index to a special character name is picked out of the *counter'th* position in the *funny_char_index_table*. A string comparision is performed between *funny_char_strings [funny_char_index_table [counter]]* and the special character name, in our example **pl**, and if it matches, then *troff* refers to this character as (96 + *counter*). When it wants to determine whether a specific font supports this character, it looks in *font_index_table[(96+counter)]*, (see below), to see whether there is a 0, meaning the character does not appear in this font, or number, which is the index into the *widths*, *kerning*, and *codes* tables.

Notice that since a value of 0 in the *font_index_table* indicates that a character does not exist, the 0th element of the *width*, *kerning*, and *codes* arrays are not used. For this reason the 0th element of the *width* array can be used for a special purpose, defining the width of a space for a font. Normally a space is defined by *troff* to be 1/3 of the width of the \(em character, but if the 0th element of the *width* array is non-zero, then that value is used for the width of a space.

FILES

/usr/lib/font/dev{X}/DESC.out description file for phototypesetter X
/usr/lib/font/dev{X}/{font}.out font description files for phototypesetter X

SEE ALSO

troff(1), troff(5).

NAME

greek – graphics for the extended TTY-37 type-box

SYNOPSIS

cat /usr/pub/greek [| **greek** –Tterminal]

DESCRIPTION

Greek gives the mapping from ASCII to the "shift-out" graphics in effect between **SO** and **SI** on TELETYPE® Model 37 terminals equipped with a 128-character type-box. These are the default greek characters produced by *nroff*(1). The filters of *greek*(1) attempt to print them on various other terminals. The file contains:

alpha	α	A	beta	β	B	gamma	γ	\
GAMMA	Γ	G	delta	δ	D	DELTA	Δ	W
epsilon	ϵ	S	zeta	ζ	Q	eta	η	N
THETA	Θ	T	theta	θ	O	lambda	λ	L
LAMBDA	Λ	E	mu	μ	M	nu	ν	@
xi	ξ	X	pi	π	J	PI	Π	P
rho	ρ	K	sigma	σ	Y	SIGMA	Σ	R
tau	τ	I	phi	ϕ	U	PHI	Φ	F
psi	ψ	V	PSI	Ψ	H	omega	ω	C
OMEGA	Ω	Z	nabla	∇	[	not	$\neg$	_
partial	∂	]	integral	$\int$	^			

FILES

/usr/pub/greek

SEE ALSO

300(1), 450(1), 4014(1), greek(1), hp(1), nroff(1), tc(1).

NAME

man − macros for formatting entries in this manual

SYNOPSIS

nroff −man files

troff −man [**−rs1**] files

DESCRIPTION

These *troff*(1) macros are used to lay out the format of the entries of this manual. A skeleton entry may be found in the file **/usr/man/u_man/man0/skeleton**. These macros are used by the *man*(1) command.

The default page size is 8.5″×11″, with a 6.5″×10″ text area; the **−rs1** option reduces these dimensions to 6″×9″ and 4.75″×8.375″, respectively; this option (which is *not* effective in *nroff*(1)*)* also reduces the default type size from 10-point to 9-point, and the vertical line spacing from 12-point to 10-point. The **−rV2** option may be used to set certain parameters to values appropriate for certain Versatec printers: it sets the line length to 82 characters, the page length to 84 lines, and it inhibits underlining; this option should not be confused with the **−Tvp** option of the *man*(1) command, which is available at some UNIX system sites.

Any *text* argument below may be one to six "words". Double quotes (**""**) may be used to include blanks in a "word". If *text* is empty, the special treatment is applied to the next line that contains text to be printed. For example, **.I** may be used to italicize a whole line, or **.SM** followed by **.B** to make small bold text. By default, hyphenation is turned off for *nroff*(1), but remains on for *troff*(1).

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, e.g., **.I**, **.RB**, **.SM**. Tab stops are neither used nor set by any macro except **.DT** and **.TH**.

Default units for indents *in* are ens. When *in* is omitted, the previous indent is used. This remembered indent is set to its default value (7.2 ens in *troff*(1), 5 ens in *nroff*this corresponds to 0.5″ in the default page size) by **.TH**, **.P**, and **.RS**, and restored by **.RE**.

.TH *t s c n* Set the title and entry heading; *t* is the title, *s* is the section number, *c* is extra commentary, e.g., "local", *n* is new manual name. Invokes **.DT** (see below).

.SH *text* Place subhead *text*, e.g., **SYNOPSIS**, here.

.SS *text* Place sub-subhead *text*, e.g., **Options**, here.

.B *text* Make *text* bold.

.I *text* Make *text* italic.

.SM *text* Make *text* 1 point smaller than default point size.

.RI *a b* Concatenate roman *a* with italic *b*, and alternate these two fonts for up to six arguments. Similar macros alternate between any two of roman, italic, and bold:

.IR .RB .BR .IB .BI

.P Begin a paragraph with normal font, point size, and indent. **.PP** is a synonym for **.P**.

.HP *in* Begin paragraph with hanging indent.

.TP *in* Begin indented paragraph with hanging tag. The next line that contains text to be printed is taken as the tag. If the tag does not fit, it is printed on a separate line.

.IP *t in* Same as **.TP** *in* with tag *t*; often used to get an indented paragraph without a tag.

.RS *in* Increase relative indent (initially zero). Indent all output an extra *in* units from the current left margin.

.RE *k* Return to the *k*th relative indent level (initially, *k*=1; *k*=0 is equivalent to *k*=1); if *k* is omitted, return to the most recent lower indent level.

.PM *m* Produces proprietary markings; where *m* may be **P** for **PRIVATE**, **N** for **NOTICE**, **BP** for **BELL LABORATORIES PROPRIETARY**, or **BR** for **BELL LABORATORIES RESTRICTED**.

.DT Restore default tab settings (every 7.2 ens in *troff*(1), 5 ens in *nroff*(1)).

.PD *v* Set the interparagraph distance to *v* vertical spaces. If *v* is omitted, set the interparagraph distance to the default value (0.4v in *troff*(1), 1v in *nroff*(1)).

The following *strings* are defined:

***R** ® in *troff*(1), **(Reg.)** in *nroff*.

***S** Change to default type size.

***(Tm** Trademark indicator.

The following *number registers* are given default values by **.TH**:

IN Left margin indent relative to subheads (default is 7.2 ens in *troff*(1), 5 ens in *nroff*(1)).

LL Line length including **IN**.

PD Current interparagraph distance.

CAVEATS

In addition to the macros, strings, and number registers mentioned above, there are defined a number of *internal* macros, strings, and number registers. Except for names predefined by *troff (1)* and number registers **d**, **m**, and **y**, all such internal names are of the form *XA*, where *X* is one of **)**, **]**, and **}**, and *A* stands for any alphanumeric character.

If a manual entry needs to be preprocessed by *eqn*(1) (or *neqn*), and/or *tbl*(1), it must begin with a special line (described in *man*(1)), causing the *man* command to invoke the appropriate preprocessor(s).

The programs that prepare the Table of Contents and the Permuted Index for this Manual assume the *NAME* section of each entry consists of a single line of input that has the following format:

name[, name, name ...] \− explanatory text

The macro package increases the inter-word spaces (to eliminate ambiguity) in the *SYNOPSIS* section of each entry.

The macro package itself uses only the roman font (so that one can replace, for example, the bold font by the constant-width font (**CW**). Of course, if the input text of an entry contains requests for other fonts (e.g., **.I**, **.RB**, **\fI**), the corresponding fonts must be mounted.

FILES

/usr/lib/tmac/tmac.an
/usr/lib/macros/cmp.n.[dt].an
/usr/lib/macros/ucmp.n.an
/usr/man/[uap]_man/man0/skeleton

SEE ALSO

ocw(1), eqn(1), man(1), nroff(1), tbl(1), tc(1), troff(1).

BUGS

If the argument to **.TH** contains *any* blanks and is *not* enclosed by double quotes (**""**), there will be strange irregular dots on the output.

NAME

math − math functions and constants

SYNOPSIS

#include <math.h>

DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various functions in the C Library (Section 3C) that return floating-point values.

It defines the structure and constants used by the *matherr*(3M) error-handling mechanisms, including the following constant used as an error-return value:

HUGE	The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E	The base of natural logarithms (e).
M_LOG2E	The base-2 logarithm of e.
M_LOG10E	The base-10 logarithm of e.
M_LN2	The natural logarithm of 2.
M_LN10	The natural logarithm of 10.
M_PI	π, the ratio of the circumference of a circle to its diameter. (There are also several fractions of π, its reciprocal, and its square root.)
M_SQRT2	The positive square root of 2.
M_SQRT1_2	The positive square root of 1/2.

For the definitions of various machine-dependent "constants," see the description of the <*values.h*> header file.

FILES

/usr/include/math.h

SEE ALSO

intro(3), matherr(3M), values(5).

NAME

mm − the MM macro package for formatting documents

SYNOPSIS

mm [options] [files]

nroff −mm [options] [files]

nroff −cm [options] [files]

mmt [options] [files]

troff −mm [options] [files]

DESCRIPTION

This package provides a formatting capability for a very wide variety of documents. It is the standard package used by the BTL typing pools and documentation centers. The manner in which a document is typed in and edited is essentially independent of whether the document is to be eventually formatted at a terminal or is to be phototypeset. See the references below for further details.

The **−mm** option causes *nroff*(1) and *troff*(1) to use the non-compacted version of the macro package, while the **−cm** option results in the use of the compacted version, thus speeding up the process of loading the macro package.

FILES

/usr/lib/tmac/tmac.m	pointer to the non-compacted version of the package
/usr/lib/macros/mm[nt]	non-compacted version of the package
/usr/lib/macros/cmp.n.[dt].m	compacted version of the package
/usr/lib/macros/ucmp.n.m	initializers for the compacted version of the package

SEE ALSO

mm(1), mmt(1), nroff(1), troff(1).

NAME

mosd – the OSDD adapter macro package for formatting documents

SYNOPSIS

osdd [options] [files]

mm **–mosd** [options] [files]

nroff **–mm** **–mosd** [options] [files]

nroff **–cm** **–mosd** [options] [files]

mmt **–mosd** [options] [files]

troff **–mm** **–mosd** [options] [files]

DESCRIPTION

The OSDD adapter macro package is a tool used in conjunction with the MM macro package to prepare Operations Systems Deliverable Documentation. Many of the OSDD Standards are different from the default format provided by MM. The OSDD adapter package sets the appropriate MM options for automatic production of the OSDD Standards. The OSDD adapter package also generates the correct OSDD page headers and footers, heading styles, Table of Contents format, etc.

OSDD document (input) files are prepared with the MM macros. Additional information which must be given at the beginning of the document file is specified by the following string definitions:

```
.ds H1 document-number
.ds H2 section-number
.ds H3 issue-number
.ds H4 date
.ds H5 rating
```

The *document-number* should be of the standard 10-character format. The words "Section" and "Issue" should not be included in the string definitions; they will be supplied automatically when the document is printed. For example:

```
.ds H1 OPA–1P135–01
.ds H2 4
.ds H3 2
```

automatically produces

```
OPA-1P135-01
Section 4
Issue 2
```

as the document page header. Quotation marks are not used in string definitions.

If certain information is not to be included in a page header, then the string is defined as null; e.g.,

```
.ds H2
```

means that there is no *section-number*.

The OSDD Standards require that the *Table of Contents* be numbered beginning with *Page 1*. By default, the first page of text will be numbered *Page 2*. If the *Table of Contents* has more than one page, for example *n*, then either **–rP***n*+*1* must be included as a command line option or **.nr P n** must be included in the document file. For example, if the *Table of Contents* is four pages then use **–rP5** on the command line or **.nr P 4** in the document file.

The OSDD Standards require that certain information such as the document *rating* appear on the *Document Index* or on the *Table of Contents* page if there is no index. By default, it is assumed that an index has been prepared

separately. If there is no index, the following must be included in the document file:

.nr Di 0

This will ensure that the necessary information is included on the *Table of Contents* page.

The OSDD Standards require that all numbered figures be placed at the end of the document. The **.Fg** macro is used to produce full page figures. This macro produces a blank page with the appropriate header, footer, and figure caption. Insertion of the actual figure on the page is a manual operation. The macro usage is

.Fg page-count "figure caption"

where *page-count* is the number of pages required for a multi-page figure (default 1 page).

The **.Fg** macro cannot be used within the document unless the final **.Fg** in a series of figures is followed by a **.SK** macro to force out the last figure page.

The *Table of Contents* for OSDD documents (see Figure 4 in Section 4.1 of the OSDD Standards) is produced with:

.Tc
System Type
System Name
Document Type
.Td

The **.Tc/.Td** macros are used instead of the **.TC** macro from MM.

The **.PM** macro may be used to generate proprietary markings – see the MM document for legal styles.

The **.P** macro is used for paragraphs. The **Np** register is set automatically to indicate the paragraph numbering style. It is very important that the **.P** macro be used correctly. All paragraphs (including those immediately following a **.H** macro) must use a **.P** macro. Unless there is a **.P** macro, there will not be a number generated for the paragraph. Similarly, the **.P** macro should not be used for text which is not a paragraph. The **.SP** macro may be appropriate for these cases, e.g., for "paragraphs" within a list item.

The page header format is produced automatically in accordance with the OSDD Standards. The OSDD Adapter macro package uses the **.TP** macro for this purpose. Therefore the **.TP** macro normally available in MM is not available for users.

FILES

/usr/lib/tmac/tmac.osd

SEE ALSO

mm(1), mmt(1), nroff(1), troff(1), mm(5).

NAME

mptx – the macro package for formatting a permuted index

SYNOPSIS

nroff –mptx [options] [files]

troff –mptx [options] [files]

DESCRIPTION

This package provides a definition for the **.xx** macro used for formatting a permuted index as produced by *ptx*(1). This package does not provide any other formatting capabilities such as headers and footers. If these or other capabilities are required, the *mptx* macro package may be used in conjuction with the *MM* macro package. In this case, the **–mptx** option must be invoked *after* the **–mm** call. For example:

```
nroff –cm –mptx file
```

or

```
mm –mptx file
```

FILES

/usr/lib/tmac/tmac.ptx	pointer to the non-compacted version of the package
/usr/lib/macros/ptx	non-compacted version of the package

SEE ALSO

mm(1), nroff(1), ptx(1), troff(1), mm(5).

NAME

mv – a troff macro package for typesetting viewgraphs and slides

SYNOPSIS

mvt [**–a**] [options] [files]

troff [**–a**] [**–rX1**] **–mv** [options] [files]

DESCRIPTION

This package makes it easy to typeset viewgraphs and projection slides in a variety of sizes. A few macros (briefly described below) accomplish most of the formatting tasks needed in making transparencies. All of the facilities of *troff*(1), *eqn*(1), and *tbl*(1) are available for more difficult tasks.

The output can be previewed on most terminals, and, in particular, on the TEKTRONIX 4014. For this device, specify the **–rX1** option (this option is automatically specified by the *mvt* command–q.v.–when that command is invoked with the **–T4014** option). To preview output on other terminals, specify the **–a** option.

The available macros are:

.VS [*n*] [*i*] [*d*] Foil-start macro; foil size is to be 7″×7″; *n* is the foil number, *i* is the foil identification, *d* is the date; the foil-start macro resets all parameters (indent, point size, etc.) to initial default values, except for the values of *i* and *d* arguments inherited from a previous foil-start macro; it also invokes the **.A** macro (see below).

The naming convention for this and the following eight macros is that the first character of the name (**V** or **S**) distinguishes between viewgraphs and slides, respectively, while the second character indicates whether the foil is square (**S**), small wide (**w**), small high (**h**), big wide (**W**), or big high (**H**). Slides are "skinnier" than the corresponding viewgraphs: the ratio of the longer dimension to the shorter one is larger for slides than for viewgraphs. As a result, slide foils can be used for viewgraphs, but not vice versa; on the other hand, viewgraphs can accommodate a bit more text.

.Vw [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″ wide × 5″ high.
.Vh [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 5″×7″.
.VW [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×5.4″.
.VH [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×9″.
.Sw [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×5″.
.Sh [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 5″×7″.
.SW [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×5.4″.
.SH [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×9″.

.A [*x*] Place text that follows at the first indentation level (left margin); the presence of *x* suppresses the ½ line spacing from the preceding text.

.B [*m* [*s*]] Place text that follows at the second indentation level; text is preceded by a mark; *m* is the mark (default is a large bullet); *s* is the increment or decrement to the point size of the mark with respect to the *prevailing* point size (default is 0); if *s* is 100, it causes the point size of the mark to be the same as that of the *default* mark.

.C [*m* [*s*]] Same as **.B**, but for the third indentation level; default mark is a dash.

.D [*m* [*s*]] Same as **.B**, but for the fourth indentation level; default mark is a small bullet.

.T *string* *String* is printed as an over-size, centered title.

.I [*in*] [*a* [*x*]] Change the current text indent (does not affect titles); *in* is the indent (in inches unless dimensioned, default is 0); if *in* is signed, it is an increment or decrement; the presence of *a* invokes the **.A** macro (see below) and passes *x* (if any) to it.

.S [*p*] [*l*] Set the point size and line length; *p* is the point size (default is "previous"); if *p* is 100, the point size reverts to the *initial* default for the current foil-start macro; if *p* is signed, it is an increment or decrement (default is 18 for **.VS**, **.VH**, and **.SH**, and 14 for the other foil-start macros); *l* is the line length (in inches unless dimensioned; default is 4.2″ for **.Vh**, 3.8″ for **.Sh**, 5″ for **.SH**, and 6″ for the other foil-start macros).

.DF *n f* [*n f* ...] Define font positions; may not appear within a foil's input text (i.e., it may only appear after all the input text for a foil, but before the next foil-start macro); *n* is the position of font *f*; up to four "*n f*" pairs may be specified; the first font named becomes the *prevailing* font; the initial setting is (**H** is a synonym for **G**):

.DF 1 H 2 I 3 B 4 S

.DV [*a*] [*b*] [*c*] [*d*] Alter the vertical spacing between indentation levels; *a* is the spacing for **.A**, *b* is for **.B**, *c* is for **.C**, and *d* is for **.D**; all non-null arguments must be dimensioned; null arguments leave the corresponding spacing unaffected; initial setting is:

.DV .5v .5v .5v 0v

.U *str1* [*str2*] Underline *str1* and concatenate *str2* (if any) to it.

The last four macros in the above list do not cause a break; the **.I** macro causes a break only if it is invoked with more than one argument; all the other macros cause a break.

The macro package also recognizes the following upper-case synonyms for the corresponding lower-case *troff* requests:

.AD .BR .CE .FI .HY .NA .NF .NH .NX .SO .SP .TA .TI

The **Tm** string produces the trademark symbol.

The input tilde (~) character is translated into a blank on output.

See the user's manual cited below for further details.

FILES

/usr/lib/tmac/tmac.v
/usr/lib/macros/vmca

SEE ALSO

eqn(1), mmt(1), tbl(1), troff(1).

BUGS

The **.VW** and **.SW** foils are meant to be 9″ wide by 7″ high, but because the typesetter paper is generally only 8″ wide, they are printed 7″ wide by 5.4″ high and have to be enlarged by a factor of 9/7 before use as viewgraphs; this makes them less than totally useful.

NAME

mview – another macro package for making view graphs

SYNOPSIS

troff –mview [options] [files] | filter [options]
nroff –mview [options] [files]

DESCRIPTION

This package provides an easy-to-use facility for making view graphs in a variety of formats. Macros are provided that accomplish most of the formatting tasks needed in making transparencies. All of the facilities of *eqn*(1) and *tbl*(1) are available for more difficult tasks. The point size and vertical spacing, if not specified by the user, is determined by the amount of data on the view graph. A message is written to diagnostic output (your terminal if not redirected) specifying the point size and vertical spacing chosen for each view graph. The output can be previewed on most terminals by using *nroff*(1). The output from *troff*(1) can be directed through the device filters *photo*(1C), *stare*(1C), etc. See the reference below for further details.

FILES

/usr/lib/tmac/tmac.view
/usr/lib/macros/mview

SEE ALSO

eqn(1), nroff(1), photo(1C), prism(1C), stare(1C), tbl(1), troff(1).
MVIEW – A Set of Macro Commands for the Generation of View Graphs by J. N. Sturman, June 21, 1978, TM number 78-8231-2.

NAME

mwp – macros for formatting Circuit Descriptions

SYNOPSIS

nroff –mwp files

troff –mwp files

DESCRIPTION

This package provides formatting capability for a Circuit Description (CDs). CDs are typed in a prescribed format which includes double-column pages; specified text headings, page headings and footers, paragraph numbering, and a Table of Contents.

.cD Invokes special Circuit Description text headings and page footers.

.2C The **.2C** macro begins two-column processing. Each page contains two columns of text of equal width. Two-column format continues until a **.1C** is specified.

.1C When the **.1C** macro is invoked following two-column text, the full width text will begin one line below the last of the longest column.

.nP The **.nP** macro produces numbered paragraphs. Paragraphs are numbered within second-level headings. (The number is automatically reset to 01 after each second-level heading.) The first two lines of the numbered paragraph are indented.

.cH"*'left'center'right'*"

The **.cH** macro may be used to generate column headings over each column when in two-column mode (**.2C**). **NOTE:** The command **.}A** must be typed on the line following the **.cH** in order to produce a header in the current column.

.eH The **.eH** macro cancels the effect of the **.cH** macro; that is, **.eH** turns off column headers. To change column headers, another **.cH** maybe invoked without using **.eH** to end the previous header.

.lP The **.lP** macro sums the pages in a document and adds the line "n Pages" (where n is the sum) below the page footer on the last page of a Circuit Description. The sectionalized CDs, where pages are numbered by section, the **.lP** macro sums the pages in each section and adds the line "n Pages" below the page footer on the last page of each section.

.SF"*'left'center'right'*"

The **.SF** macro provides an additional line of page-footer text. This text will appear one line below the text specified by **.PF**, if given.

FILES

/usr/lib/tmac/tmac.wp

SEE ALSO

UNIX TEXT PROCESSING REFERENCE MANUAL.

NAME

news – USENET network news article, utility files

DESCRIPTION

There are two formats of news articles: **A** and **B.** **A** format is the only format that version 1 netnews systems can read or write. Systems running the version 2 netnews can read either format and there are provisions for the version 2 netnews to write in **A** format. **A** format looks like this:

A*article-ID*
newsgroups
path
date
title
Body of article

Only version 2 netnews systems can read and write **B** format. **B** format contains two extra pieces of information: receival date and expiration date. The basic structure of a **B** format file consists of a series of headers and then the body. A header field is defined as a line with a capital letter in the 1st column and a colon somewhere on the line. Unrecognized header fields are ignored. News is stored in the same format transmitted, see "Standard for the Interchange of USENET Messages" for a full description. The following fields are among those recognized:

Header	Information
From:	*user@host.domain[.domain ...] (Full Name)*
Newsgroups:	*Newsgroups*
Message-ID:	*<Unique Identifier>*
Subject:	*descriptive title*
Date:	*Date Posted*
Date-Received:	*Date received on local machine*
Expires.	*Expiration Date*
Reply-To:	*Address for mail replies*
References:	*Article ID of article this is*
Control:	*Text of a control message*

Here is an example of an article:

Relay-Version: B 2.10 2/13/83 cbosgd.UUCP
Posting-Version: B 2.10 2/13/83 eagle.UUCP
Path: cbosgd!mhuxj!mhuxt!eagle!jerry
From: jerry@eagle.uucp (Jerry Schwarz)
Newsgroups: net.general
Subject: Usenet Etiquette -- Please Read
Message-ID: <642@eagle.UUCP>
Date: Friday, 19-Nov-82 16:14:55 EST
Followup-To: net.news
Expires: Saturday, 1-Jan-83 00:00:00 EST
Date-Received: Friday, 19-Nov-82 16:59:30 EST
Organization: Bell Labs, Murray Hill

The body of the article comes here, after a blank line.

A *sys* file line has four fields, each separated by colons:

system-name:subscriptions:flags:transmission command

Of these fields, on the *system-name* and *subscriptions* need to be present.

The *system name* is the name of the system being sent to. The *subscriptions* is the list of newsgroups to be transmitted to the system. The *flags* are a set of letters describing how the article should be transmitted. The default is B. Valid flags include A (send in A format), B (send in B format), N (use ihave/sendme protocol), U (use uux -c and the name of the stored article in a %s string).

The *transmission command* is executed by the shell with the article to be transmitted as the standard input. The default is **uux - -z -r** *sysname*!**rnews.** Some examples:

xyz:net.all
oldsys:net.all,fa.all,to.oldsys:A
berksys:net.all,ucb.all::/usr/lib/news/sendnews -b berksys:rnews
arpasys:net.all,arpa.all::/usr/lib/news/sendnews -a rnews@arpasys
old2:net.all,fa.all:A:/usr/lib/sendnews -o old2:rnews
user:fa.sf-lovers::mail user

Somewhere in a *sys* file, there must be a line for the host system. This line has no *flags* or *commands*. A # as the first character in a line denotes a comment.

The history, active, and ngfile files have one line per item.

SEE ALSO

inews(1), postnews(1), readnews(1), sendnews(8), uurec(8).

NAME

newsrc – information file for readnews and checknews

DESCRIPTION

The *.newsrc* file contains the list of previously read articles and an optional options line for *readnews(1)* and *checknews(1)*. Each newsgroup that articles have been read from has a line of the form:

newsgroup: range

The *range* is a list of the articles read. It is basically a list of no.'s separated by commas with sequential no.'s collapsed with hyphens. For instance:

general: 1-78,80,85-90
fa.info-cpm: 1-7
net.news: 1
fa.info-vax! 1-5

If the : is replaced with an ! (as in info-vax above) the newsgroup is not subscribed to and will not be shown to the user.

An options line starts with the word **options** (left-justified). Then there are the list of options just as they would be on the command line. For instance:

options −n all !fa.sf-lovers !fa.human-nets −r
options −c −r

A string of lines beginning with a space or tab after the initial options line will be considered continuation lines.

FILES

~/.newsrc options and list of previously read articles

SEE ALSO

checknews(1), readnews(1).

NAME

prof − profile within a function

SYNOPSIS

```
#define MARK
#include <prof.h>

void MARK (name)
```

DESCRIPTION

MARK will introduce a mark called *name* that will be treated the same as a function entry point. Execution of the mark will add to a counter for that mark, and program-counter time spent will be accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

Name may be any combination of up to six letters, numbers or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header file <*prof.h*> is included. This may be defined by a preprocessor directive as in the synopsis, or by a command line argument, i.e:

```
cc −p −DMARK foo.c
```

If MARK is not defined, the *MARK*(name) statements may be left in the source files containing them and will be ignored.

EXAMPLE

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with *MARK* defined on the command line, the marks are ignored.

```
#include <prof.h>

foo( )
{
        int i, j;

        .
        .
        .
        MARK(loop1);
        for (i = 0; i < 2000; i++) {
                . . .
        }
        MARK(loop2);
        for (j = 0; j < 2000; j++) {
                . . .
        }
}
```

SEE ALSO

prof(1), profil(2), monitor(3C).

NAME

regexp – regular expression compile and match routines

SYNOPSIS

```
#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include <regexp.h>

char *compile (instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;

int step (string, expbuf)
char *string, *expbuf;

extern char *loc1, *loc2, *locs;

extern int circf, sed, nbra;
```

DESCRIPTION

This page describes general-purpose regular expression matching routines in the form of *ed*(1), defined in **/usr/include/regexp.h**. Programs such as *ed*, *sed*(1), *grep*(1), *bs*(1), *expr*(1), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the "#include <regexp.h>" statement. These macros are used by the *compile* routine.

GETC()	Return the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression.
PEEKC()	Return the next character in the regular expression. Successive calls to PEEKC() should return the same character (which should also be the next character returned by GETC()).
UNGETC(*c*)	Cause the argument *c* to be returned by the next call to GETC() (and PEEKC()). No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC(). The value of the macro UNGETC(*c*) is always ignored.
RETURN(*pointer*)	This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.
ERROR(*val*)	This is the abnormal return from the *compile* routine. The argument *val* is an error number (see table below for meanings). This call should never return.

ERROR	MEANING
11	Range endpoint too large.
16	Bad number.
25	"\digit" out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\(\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \{ \}.
45	} expected after \.
46	First number exceeds second in \{ \}.
49	[ ] imbalance.
50	Regular expression overflow.

The syntax of the *compile* routine is as follows:

compile(instring, expbuf, endbuf, eof)

The first parameter *instring* is never used explicitly by the *compile* routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char *) 0) for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf*–*expbuf*) bytes, a call to ERROR(50) is made.

The parameter *eof* is the character which marks the end of the regular expression. For example, in *ed*, this character is usually a /.

Each program that includes this file must have a **#define** statement for INIT. This definition will be placed right after the declaration for the function *compile* and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for GETC(), PEEKC() and UNGETC(). Otherwise it can be used to declare external variables that might be used by GETC(), PEEKC() and UNGETC(). See the example below of the declarations taken from *grep*.

There are other functions in this file which perform actual regular expression matching, one of which is the function *step*. The call to *step* is as follows:

step(string, expbuf)

The first parameter to *step* is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter *expbuf* is the compiled regular expression which was obtained by a call of the function *compile*.

The function *step* returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to *step*. The variable set in *step* is *loc1*. This is a pointer to the first character that matched the regular expression. The variable *loc2*, which is set by the function *advance*, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, *loc1* will point to the first character of *string* and *loc2* will point to the null at the end of *string*.

Step uses the external variable *circf* which is set by *compile* if the regular expression begins with ^. If this is set then *step* will try to match the regular expression to the beginning of the string only. If more than one regular expression is to be compiled before the first is executed the value of *circf* should be saved for each compiled expression and *circf* should be set to that saved value before each call to *step*.

The function *advance* is called from *step* with the same arguments as *step*. The purpose of *step* is to step through the *string* argument and call *advance* until *advance* returns non-zero indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, *step* need not be called; simply call *advance*.

When *advance* encounters a * or \{ \} sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, *advance* will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer *locs* is equal to the point in the string at sometime during the backing up process, *advance* will break out of the loop that backs up and will return zero. This is used by *ed* and *sed* for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like **s/y*//g** do not loop forever.

The additional external variables *sed* and *nbra* are used for special purposes.

EXAMPLES

The following is an example of how the regular expression macros and calls look from *grep*:

```
#define INIT            register char *sp = instring;
#define GETC()          (*sp++)
#define PEEKC()         (*sp)
#define UNGETC(c)       (--sp)
#define RETURN(c)       return;
#define ERROR(c)        regerr()

#include <regexp.h>
...
        (void) compile(*argv, expbuf, &expbuf[ESIZE], '\0');
...
        if (step(linebuf, expbuf))
                succeed();
```

FILES

/usr/include/regexp.h

SEE ALSO

bs(1), ed(1), expr(1), grep(1), sed(1).

BUGS

The handling of *circf* is kludgy.
The actual code is probably easier to understand than this manual page.

NAME

stat – data returned by stat system call

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
```

DESCRIPTION

The system calls *stat* and *fstat* return data whose structure is defined by this include file. The encoding of the field *st_mode* is defined in this file also.

```
/*
 * Structure of the result of stat
 */

struct  stat
{
        dev_t   st_dev;
        ino_t   st_ino;
        ushort  st_mode;
        short   st_nlink;
        ushort  st_uid;
        ushort  st_gid;
        dev_t   st_rdev;
        off_t   st_size;
        time_t  st_atime;
        time_t  st_mtime;
        time_t  st_ctime;
};

#define S_IFMT   0170000 /* type of file */
#define S_IFDIR  0040000 /* directory */
#define S_IFCHR  0020000 /* character special */
#define S_IFBLK  0060000 /* block special */
#define S_IFREG  0100000 /* regular */
#define S_IFIFO  0010000 /* fifo */
#define S_ISUID  04000   /* set user id on execution */
#define S_ISGID  02000   /* set group id on execution */
#define S_ISVTX  01000   /* save swapped text even after use */
#define S_IREAD  00400   /* read permission, owner */
#define S_IWRITE 00200   /* write permission, owner */
#define S_IEXEC  00100   /* execute/search permission, owner */
```

FILES

/usr/include/sys/types.h
/usr/include/sys/stat.h

SEE ALSO

stat(2), types(5).

NAME

summarize - summarize USORS daily data and maintain files

SYNOPSIS

summarize

DESCRIPTION

Summarize is a shell script, designed for nightly invocation from the "cron" table. It calculates, using the "daysum" program, the USORS median and 90th percentiles for the day overall and for the busiest hour of the day ("busy" being defined as having the largest median). It excludes work holidays from the summary file by referring to a holiday list valid through the year 1999.

A degree of robustness was designed into the program: it checks to see that summary lines exist for the last seven days, computing any missing lines. *Summarize* also maintains files in the data directory, deleting files collected during work holidays, and deleting those that have not been modified within the past 63 days.

FILES

/usr/adm/usors/bin/summarize
/usr/adm/usors/sum/days
/usr/adm/usors/sum/holiday

SEE ALSO

dayplot(1), daysum(1), overview(1), usors(1).

BUGS

"Holiday" file must be updated in year 1999, and 4 lines within "summarize" decommented.

NAME

tabs – discussion of tab handling on UNIX™

DESCRIPTION

The following is a guide to using tabs on UNIX.

TABS AND YOUR TERMINAL

If a terminal does not have a tab key, the tab character can be produced by holding down the control (CTRL) key and depressing "i". This allows all ASCII terminals to produce a tab, although not many can actually process it. If the computer sends a tab character to the terminal and the terminal "supports" tabs, the print head or cursor will move to the next tab stop; if the terminal does not support tabs, the tab character will be ignored.

By default, UNIX assumes the terminal cannot support tabs and replaces the tab character with one or more spaces. This is accomplished by the terminal driver simulating standard tabs, which are defined to be every eight positions starting in column one.

Terminals having hardware settable tabs can be set using two commands. First, the terminal tab stops are set using the *tabs*(1) command. Second, by typing "stty tabs", UNIX will understand not to simulate tabs (see *stty*(1)). The terminal driver will then send the tab characters to the terminal (instead of blanks) and the physical tab stops set by the *tabs* command will be used.

UNIX COMMANDS AND TABS

Nroff(1)

By default, *nroff* uses multiple blanks rather than tabs in its output. The "-h" option, however, will force *nroff* to output a tab character instead of blanks wherever possible to speed output printing. This option only works with standard tabs, whether simulated by UNIX or set by the *tabs* command.

Nroff also allows the user to set logical tab stops via the ".ta" command. These tab stops are used by *nroff* to determine the placement of a field during formatting, similar to *tbl* (discussed below). What actually happens on the terminal when the user types a tab character or when UNIX sends a tab character to the terminal depends on the logical tab stops or UNIX simulating tabs.

Gath(1)

The *gath* command expands tab characters present in a file to blanks according to the format specification for that file (see *fspec*(5) and below). However, like *nroff*, *gath* does not output tabs by default. The *gath* "-h" option causes tabs to be used in output printing. This option, similar to the *nroff* "-h" option, will only work with standard tabs.

Fspec(5)

The *fspec* specification allows a user to define tab settings for text files. These specifications are interpreted by various commands, such as *gath* and *ed*(1). If physical tabs are set when using *ed* on a file containing an *fspec*, the tabs must be set to be the same as the *fspec*. *Ed* only expands tabs according to the *fspec* on output; input tabs will behave normally.

Tbl(1)

The *tbl* command uses the tab as its delimiter character; the tab simply tells it where a field starts and ends. The expansion of tabs on input does not affect the appearance of the *tbl* output, e.g., a user should not use multiple tabs to line up the input to *tbl*. The tab in this case is simply the character or delimiter used between columns.

SEE ALSO

ed(1), gath(1), nroff(1), send(1), stty(1), tabs(1), tbl(1), fspec(5)

TERM(5)

NAME

term − conventional names for terminals

DESCRIPTION

These names are used by certain commands (e.g., *tabs*(1), *man*(1) and are maintained as part of the shell environment (see *sh*(1), *profile*(4), and *environ*(5)) in the variable **$TERM**:

1520	Datamedia 1520
1620	DIABLO 1620 and others using the HyType II printer
1620−12	same, in 12-pitch mode
2621	Hewlett-Packard HP2621 series
2631	Hewlett-Packard 2631 line printer
2631−c	Hewlett-Packard 2631 line printer - compressed mode
2631−e	Hewlett-Packard 2631 line printer - expanded mode
2640	Hewlett-Packard HP2640 series
2645	Hewlett-Packard HP264n series (other than the 2640 series)
300	DASI/DTC/GSI 300 and others using the HyType I printer
300−12	same, in 12-pitch mode
300s	DASI/DTC/GSI 300s
382	DTC 382
300s−12	same, in 12-pitch mode
3045	Datamedia 3045
33	TELETYPE® Model 33 KSR
37	TELETYPE Model 37 KSR
40−2	TELETYPE Model 40/2
40−4	TELETYPE Model 40/4
4540	TELETYPE Model 4540
3270	IBM Model 3270
4000a	Trendata 4000a
4014	TEKTRONIX 4014
43	TELETYPE Model 43 KSR
450	DASI 450 (same as Diablo 1620)
450−12	same, in 12-pitch mode
735	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
sync	generic name for synchronous TELETYPE 4540-compatible terminals
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
tn1200	User Electric TermiNet 1200
tn300	User Electric TermiNet 300

Up to 8 characters, chosen from [−a−z0−9], make up a basic terminal name. Terminal sub-models and operational modes are distinguished by suffixes beginning with a −. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

Commands whose behavior depends on the type of terminal should accept arguments of the form **−T***term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **$TERM**, which, in turn, should contain *term*.

SEE ALSO

man(1), mm(1), nroff(1), sh(1), stty(1), tabs(1), tplot(1G), profile(4), environ(5).

BUGS

This is a small candle trying to illuminate a large, dark problem. Programs that ought to adhere to this nomenclature do so somewhat fitfully.

NAME

troff – description of output language

DESCRIPTION

The device-independent *troff* outputs a pure ASCII description of a typeset document. The description specifies the typesetting device, the fonts, and the point sizes of characters to be used as well as the position of each character on the page. A list of all the legal commands follows. Most numbers are denoted as *n* and are ASCII strings. Strings inside of [ ] are optional. *Troff* may produce them, but they are not required for the specification of the language. The character **\n** has the standard meaning of "newline" character. Between commands white space has no meaning. White space characters are spaces and newlines. All commands which have an arbitary length numerical parameter or word must be followed by white space. For example, the command to specify point size, **s###**, must be followed by a space or newline.

s*n* — The point size of the characters to be generated.

f*n* — The font mounted in the specified position is to be used. The number ranges from 0 to the highest font presently mounted. 0 is a special position, invoked by *troff*, but not directly accessible to the troff user. Normally fonts are mounted starting at position 1.

c*x* — Generate the character *x* at the current location on the page; *x* is a single ASCII character.

C*xyz* — Generate the special character *xyz*. The name of the character is delimited by white space. The name will be one of the special characters legal for the typesetting device as specified by the device specification found in the file *DESC*. This file resides in a directory specific for the typesetting device. (See *font*(5) and **/usr/lib/font/dev***.)

H*n* — Change the horizonal position on the page to the number specified. The number is in basic units of motions as specified by *DESC*. This is an absolute "goto".

h*n* — Add the number specified to the current horizontal position. This is a relative "goto".

V*n* — Change the vertical position on the page to the number specified (down is positive).

v*n* — Add the number specified to the current vertical position.

nnx — This is a two-digit number followed by an ASCII character. The meaning is a combination of **h***n* followed by **c***x*. The two digits *nn* are added to the current horizontal position and then the ASCII character, *x*, is produced. This is the most common form of character specification.

n*b a* — This command indicates that the end of a line has been reached. No action is required, though by convention the horizontal position is set to 0. *Troff* will specify a resetting of the *x*,*y* coordinates on the page before requesting that more characters be printed. The first number, *b*, is the amount of space before the line and the second number, *a*, the amount of space after the

line. The second number is delimited by white space.

w A w appears between words of the input document. No action is required. It is included so that one device can be emulated more easily on another device.

p*n* Begin a new page. The new page number is included in this command. The vertical position on the page should be set to 0.

{ Push the current environment, which means saving the current point size, font, and location on the page.

} Pop a saved environment.

t*xxxxx* Print the string of characters, *xxxxx*, using the natural width of each character to determine the next x coordinate. *Troff* does not currently produce this form of command. It is not recommended. The characters will probably be too close together.

.... **\n** A line beginning with a pound sign is a comment.

Dl *x y***\n** Draw a line from the current location to *x*,*y*. At the end of the drawing operation the current location will be *x*,*y*.

Dc *d***\n** Draw a circle of diameter *d* with the leftmost edge being at the current location (x, y). The current location after drawing the circle will be x+*d*,y, the rightmost edge of the circle.

De *dx dy***\n** Draw an ellipse with the specified axes. *dx* is the axis in the x direction and *dy* is the axis in the y direction. The leftmost edge of the ellipse will be at the current location. After drawing the ellipse the current location will be x+*dx*,y.

Da *x y r***\n** Draw a counterclockwise arc from the current location to *x*,*y* using a circle of radius *r* . The current location after drawing the arc will be *x*,*y*.

D~ *x y x y*...**\n** Draw a spline curve (wiggly line) between each of the *x*,*y* coordinate pairs starting at the current location. The final location will be the final *x*,*y* pair of the list. Currently there may be no more than 36 *x*,*y* pairs to this command.

x i[nit]\n Initialize the typesetting device. The actions required are dependent on the device. An **init** command will always occur before any output generation is attempted.

x T *device***\n** The name of the typesetter is *device*. This is the same as the argument to the **−T** option. The information about the typesetter will be found in the directory **/usr/lib/font/dev**{*device*}.

x r[es] *n h v***\n** The resolution of the typesetting device in increments per inch is *n*. Motion in the horizontal direction can take place in units of *h* basic increments. Motion in the vertical direction can take place in units of *v* basic increments. For example, the APS-5 typesetter has a basic resolution of 723 increments per inch and can move in either direction in 723rds of an inch. Its specification is:
x res 723 1 1

x p[ause]\n Pause. Cause the current page to finish but do not relinquish the typesetter.

x s[top]\n Stop. Cause the current page to finish and then relinquish the typesetter. Perform any shutdown and bookkeeping procedures required.

x t[railer]\n Generate a trailer. On some devices no operation is performed.

x f[ont] *n name*\n Load the font *name* into position *n*.

x H[eight] *n*\n Set the character height to *n* points. This causes the letters to be elongated or shortened. It does not affect the width of a letter.

x S[lant] *n*\n Set the slant to *n* degrees. Only some typesetters can do this and not all angles are supported.

NAME

types – primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
typedef  struct { int r[1]; } *        physadr;
typedef  long                  daddr_t;
typedef  char *                caddr_t;
typedef  unsigned int          uint;
typedef  unsigned short        ushort;
typedef  ushort                ino_t;
typedef  short                 cnt_t;
typedef  long                  time_t;
typedef  int                   label_t[10];
typedef  short                 dev_t;
typedef  long                  off_t;
typedef  long                  paddr_t;
typedef  long                  key_t;
```

The form *daddr_t* is used for disk addresses except in an i-node on disk, see *fs*(4). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

SEE ALSO

fs(4).

NAME

values – machine-dependent values

SYNOPSIS

#include <values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS(*type*)	The number of bits in a specified type (e.g., int).
HIBITS	The value of a short integer with only the high-order bit set (in most implementations, 0x8000).
HIBITL	The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).
HIBITI	The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).
MAXSHORT	The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767).
MAXLONG	The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647).
MAXINT	The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).
MAXFLOAT, LN_MAXFLOAT	The maximum value of a single-precision floating-point number, and its natural logarithm.
MAXDOUBLE, LN_MAXDOUBLE	The maximum value of a double-precision floating-point number, and its natural logarithm.
MINFLOAT, LN_MINFLOAT	The minimum positive value of a single-precision floating-point number, and its natural logarithm.
MINDOUBLE, LN_MINDOUBLE	The minimum positive value of a double-precision floating-point number, and its natural logarithm.
FSIGNIF	The number of significant bits in the mantissa of a single-precision floating-point number.
DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number.

FILES

/usr/include/values.h

SEE ALSO

intro(3), math(5).

NAME

varargs – handle variable argument list

SYNOPSIS

#include <varargs.h>

va_alist

va_dcl

void va_start(pvar)
va_list pvar;

type **va_arg(pvar,** *type***)**
va_list pvar;

void va_end(pvar)
va_list pvar;

DESCRIPTION

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists (such as *printf*(3S)) but do not use *varargs* are inherently nonportable, as different machines use different argument-passing conventions.

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for *va_alist*. No semicolon should follow *va_dcl*.

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize *pvar* to the beginning of the list.

va_arg will return the next argument in the list pointed to by *pvar*. *Type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by *va_start ... va_end*, are possible.

EXAMPLE

This example is a possible implementation of *execl*(2).

```
#include <varargs.h>
#define MAXARGS     100

/*      execl is called by
                execl(file, arg1, arg2, ..., (char *)0);
*/
execl(va_alist)
va_dcl
{
        va_list ap;
        char *file;
        char *args[MAXARGS];
        int argno = 0;

        va_start(ap);
        file = va_arg(ap, char *);
        while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
                ;
        va_end(ap);
        return execv(file, args);
}
```

SEE ALSO

exec(2), printf(3S).

BUGS

It is up to the calling routine to specify how many arguments there are, since it is not always possible to determine this from the stack frame. For example, *execl* is passed a zero pointer to signal the end of the list. *Printf* can tell how many arguments are there by the format.

It is non-portable to specify a second argument of *char*, *short*, or *float* to *va_arg*, since arguments seen by the called function are not *char*, *short*, or *float*. C converts *char* and *short* arguments to *int* and converts *float* arguments to *double* before passing them to a function.